VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VIRTUALIZACE FITKITU

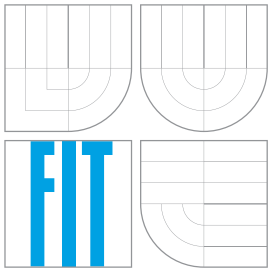
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK VAVRUŠA

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VIRTUALIZACE FITKITU

FITKIT VIRTUALIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MAREK VAVRUŠA
AUTHOR

VEDOUCÍ PRÁCE — ING. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2011

Abstrakt

Práce se zabývá komplexní virtualizací výukové platformy FITkit.
První část bakalářské práce popisuje rozšíření knihovny libkitclient o podporu virtuálních zařízení a návrh protokolu pro sdílení platformy FITkit v IP sítích. Protokol je rozšířen o podporu automatického vyhledávání zařízení v místní síti pomocí protokolu UDP multicast. Druhá část práce se zabývá návrhem architektury pro vzdálený překlad aplikací pro platformu FITkit a její implementací v programech fcmake a QDevKit. Poslední část práce shrnuje dosažené výsledky a představuje další možnosti rozšíření.

Abstract

Thesis presents a method of FITkit hardware platform virtualization. First part describes an extension of the libkitclient library to support virtual devices and discusses a protocol design for FITkit device sharing in the IP networks. The protocol is extended to offer an automatic device discovery in local network based on UDP multicast. The second part of the thesis presents an architecture for a remote compilation of FITkit platform binaries and the implementation in the fcmake and QDevKit. The last chapter summarizes implemented tools and presents possible future extensions.

Klíčová slova

FITkit, QDevKit, libkitclient, virtualizace, síť, TCP/IP, protokol, ASN.1, X.680, X.690, UDP, TCP, multicast

Keywords

FITkit, QDevKit, libkitclient, virtualization, network, TCP/IP, protocol, ASN.1, X.680, X.690, UDP, TCP, multicast

Citace

Marek Vavruša: Virtualizace FITkitu, bakalářská práce, Brno, FIT VUT v Brně, 2011

Virtualizace FITkitu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka

.....
Marek Vavruša
17. května 2011

Poděkování

Chtěl poděkovat svému vedoucímu Ing. Zdeňku Vašíčkovi za obětavé a odborné vedení v průběhu celého bakalářského studia.

Obsah

Kapitola 1

Úvod

Hlavní motivací bakalářské práce zabývající se možnostmi virtualizace platformy FITkit je problematická a často náročná instalace softwarového vybavení potřebného pro překlad aplikací pro FITkit. Další přínos lze zpatřit v oblasti propagace. Díky virtualizci mají případní zájemci možnost si vyzkoušet práci s platformou vzdáleně, aniž by ji museli fyzicky vlastnit či instalovat potřebný software. V rámci projektu, který se zabývá realizací virtuální laboratoře, můžeme s navrženým přístupem ušetřit značné množství energie, neboť všechna zařízení mohou být připojena k centrální stanici, která se bude starat o jejich zpřístupnění klientským stanicím. Mimo to sdílení zařízení v IP sítích usnadní studentům krátkodobé půjčovaní FITkitu bez nutnosti ho fyzicky přepojovat. Problém náročné instalace softwarového vybavení řeší virtualizace překladového prostředí, která dovoluje překlad aplikací bez nutnosti instalovat nástroje lokálně.

Náplní této bakalářské práce je rozšířit softwarové vybavení platformy FITkit o možnost spravovat virtuální zařízení a implementovat jak virtuální zařízení komunikující s fyzickými, tak architekturu pro vyhledávání a sledování stavu těchto zařízení.

Práce je členěna následovně. První kapitola se zabývá úvodem do řešené problematiky, jsou zde popsány vývojové nástroje používané pro usnadnění práce s platformou FITkit, především pak knihovna *libkitclient* a aplikace QDevKit. Druhá a třetí kapitola se věnuje specifikaci a návrhu komunikačního protokolu, který je použit pro sdílení zařízení v síti. Protokol implementuje standard X.680 ASN.1[5] s kódováním X.690 BER[6] a využívá protokol TCP pro zajištění spolehlivého přenosu dat.

Jako rozšíření je zde navržen a implementován jednoduchý protokol pro automatické vyhledávání zařízení v místních IP sítích. Závěrem této části práce je podpora obou protokolů a praktické využití v programu QDevKit. Čtvrtá kapitola se zabývá překladem aplikací pro platformu FITkit na vzdáleném stroji bez nutnosti instalace dalšího softwarového vybavení. V této části je popsána realizace zabezpečené metody poskytující přístup k překladovému serveru jak v rámci sítě VUT tak i mimo. V závěru jsou shrnuty dosažené výsledky a navržena možná rozšíření.

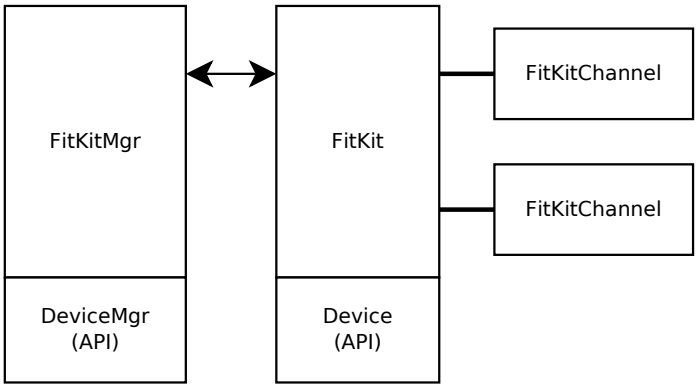
Kapitola 2

Nástroje pro FITkit

Pro platformu FITkit existuje softwarové vybavení pro sériovou komunikaci, programování mikroprocesoru a rekonfiguraci hradlového pole. Základní knihovnu pro platformově nezávislou komunikaci se zařízením FITkit představuje *libkitclient*. Knihovna je napsána v jazyce C++, ale současně je pro ni vytvořeno rozhraní v jazyce Python. Tohoto rozhraní je využito např. v aplikaci FKFlash pro programování platformy FITkit a terminálové aplikaci FKTerm. Knihovna *libkitclient* představuje základ pro komunikaci v grafické terminálové aplikaci QDevKit. Pro překlad aplikací se však používá překladač MSP-GCC a vývojové prostředí Xilinx ISE pro generování konfiguračního řetězce.

2.1 Knihovna libkitclient

Knihovna *libkitclient* se skládá z abstrakce rozhraní pro platformově nezávislou komunikaci se zařízením FITkit a správce připojených zařízení. Zařízení FITkit je reprezentováno třídou `FitKit` a obsahuje právě dva komunikační kanály třídy `FitKitChannel`. Správce zařízení představuje dle schématu 2.1 třída `FitKitMgr` implementující šablonu `DeviceMgr`. Ze své podstaty správce vytváří a obsluhuje právě jeden typ zařízení. V prostředí Microsoft Windows jsou tyto třídy implementovány pomocí knihovny *D2XX*, oficiální podporované knihovny pro přímou komunikaci s převodníky FTDI. Pro GNU/Linux je komunikace implementována na základě jednodušší knihovny *libftdi*.



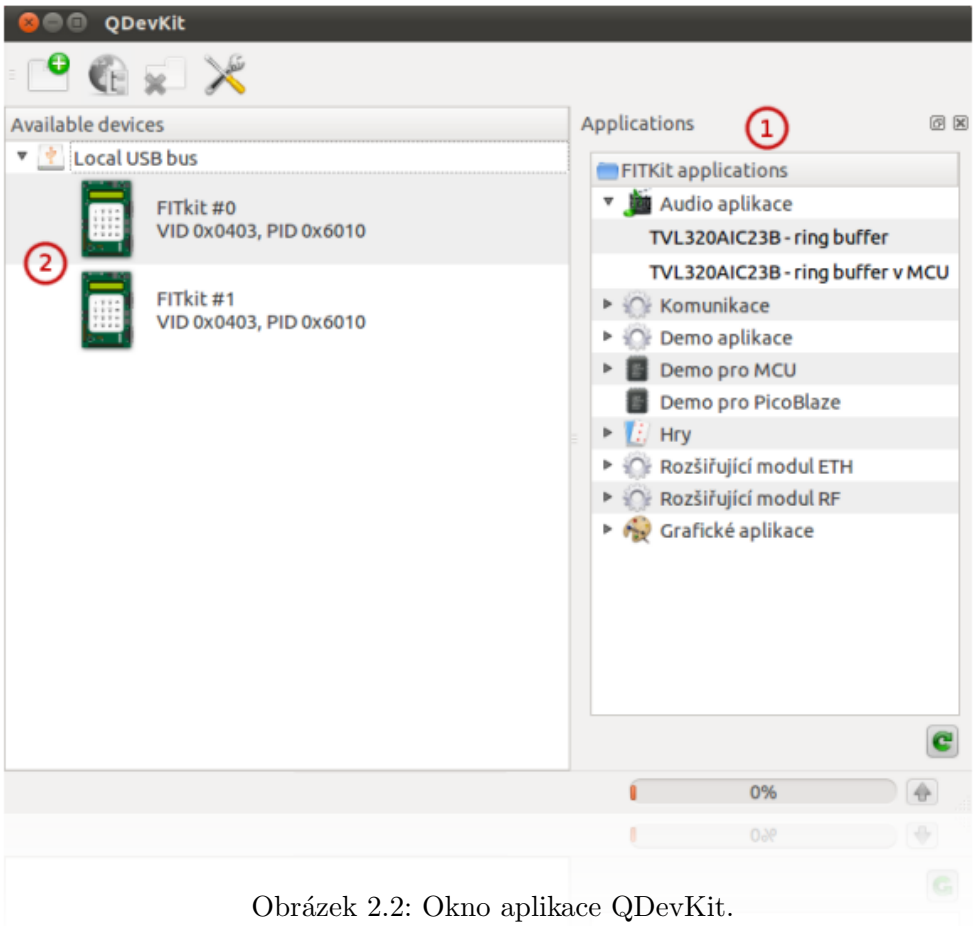
Obrázek 2.1: Blokové schéma správce zařízení `FitKitMgr`.

2.2 FCMake

Aplikace FCMake slouží především pro generování překladových souborů *Makefile* pro aplikace na platformu FITkit, které jsou popsány definičními soubory v jazyce XML. Definiční soubory obsahují jak metadata o aplikaci, tak seznam zdrojových souborů a závislostí konkrétní aplikace. Sdílená knihovna aplikace FCMake poskytuje rozhraní pro interpretaci těchto souborů, které je využito pro seznam aplikací v grafické terminálové aplikaci QDevKit. Aplikace je napsána v jazyce C++ s využitím knihovny Qt[11].

2.3 QDevKit

Grafická terminálová aplikace QDevKit (obrázek 2.2) napsaná v jazyce C++ umožňuje mimo sériovou komunikaci se zařízením FITkit ve formě terminálu také programování mikrokontroleru i rekonfiguraci hradlového pole pomocí komponenty FKFlash. Zároveň přehledně zobrazuje **(1)** dostupné aplikace pro platformu FITkit ve formě stromu a umožňuje jejich aktualizaci z centrálního repozitáře projektu FITkit. Aplikace pro platformu FITkit je možné překládat přímo v rámci aplikace, pokud jsou nainstalovány nástroje pro překlad (MSP-GCC a Xilinx ISE). QDevKit také umožňuje spouštět simulaci pokud je nalezen simulační software ModelSim. Zařízení FITkit jsou v aplikaci reprezentována **(2)** seznamem ikon s podrobnějšími údaji a otevřené komunikační kanály ve formě záložek s terminálem. Platformově nezávislé grafické rozhraní je implementováno pomocí knihovny Qt[11].



Obrázek 2.2: Okno aplikace QDevKit.

Kapitola 3

Podpora virtuálních zařízení

Knihovna *libkitclient* byla původně navržena pro práci s jedním typem zařízení (třída `FitKit`) a protože vytváření instancí nových zařízení včetně implementačně závislých operací bylo součástí konkrétní třídy správce zařízení (třídy `FitKitMgr`), nebylo jednoduše možné obsluhovat virtuální zařízení spolu s fyzickými. Zároveň ale bylo nanejvýš vhodné pokud možno zachovat kompatibilitu na úrovni zdrojového kódu se starší verzí knihovny *libkitclient*.

Kompatibility na úrovni zdrojových kódů je částečně dosaženo vytvořením čistě virtuálního API pro práci se zařízením a modulární architektury správy zařízení.

3.1 Změny ve správě zařízení

V původní verzi *libkitclient* sloužila třída `DeviceMgr` jako šablona pro implementaci obsluhy jednoho typu zařízení, např. `FitKitMgr`. Z toho důvodu byla třída `DeviceMgr` nově implementována jako plně autonomní bez nutnosti dalšího rozšiřování a pro implementaci konkrétních potřeb daného typu zařízení byl zaveden systém modulů. Každý modul implementuje rozhraní třídy `DeviceBackend` a obsluhuje právě jeden typ zařízení.

Povinné rozhraní modulu

Povinné rozhraní modulu třídy `DeviceBackend` zahrnuje metody pro:

Vytvoření konkrétní instance zařízení, která musí implementovat rozhraní třídy `Device`.
Jedná se o metodu `Device* create(DeviceData*)`.

Inicializaci zařízení, např. pro nastavení stavu nebo inicializaci dalších prostředků.
Jedná se o metodu `bool initialize(Device *device)`.

Kontrolu komunikačních kanálů připojeného zařízení.
Jedná se o metodu `bool isReady(DeviceData*)`.

Aktualizaci seznamu všech připojených zařízení, které daný modul spravuje.
Jedná se o metodu `void update(DeviceData::List&)`.

Nepovinné rozhraní modulu

Zároveň může každý modul implementovat i nepovinné metody pro obsluhu událostí:

Zavedení modulu, zde je možné především inicializovat potřebné prostředky pro hledání náležících zařízení. Jedná se o metodu `bool start()`.

Zastavení modulu, zejména pro uvolnění zabraných prostředků.
Jedná se o metodu `bool stop()`.

Obsazení volného zařízení pro reakci na změnu stavu zařízení.
Jedná se o metodu `void deviceAcquired(int)`.

Uvolnění obsazeného zařízení.
Jedná se o metodu `void deviceReleased(int)`.

Nalezení nového zařízení.
Jedná se o metodu `void deviceFound(int)`.

Odpojení existujícího zařízení z důvodu chyby v komunikaci nebo vyžádaného odpojení zařízení. Jedná se o metodu `void deviceLost(int)`.

Původní třídu `FitKitMgr` nyní implementuje modul `FTDIBackend`.

3.2 Změny v komunikaci se zařízením

Analogicky ke správě zařízení došlo k nahrazení třídy `FitKit` pro komunikaci se zařízením a třídy `FitKitChannel` pro komunikaci s jedním kanálem zařízení. Se všemi zařízeními se nyní pracuje přes čistě virtuální rozhraní `Device` a s kanály přes rozhraní `IOChannel`. Metody pro práci se zařízením i kanály zůstaly zachovány.

Obdobně jako v případě správy zařízení, moduly implementují funkcionalitu jednotlivého typu zařízení který spravují. Funkcionalitu původní třídy `FitKit` zastupuje třída `FTDIDevice` a třídu `FitKitChannel` nyní implementuje `FTDIChannel`. Hlavní rozdíl tedy představuje oddělení veřejného API a implementace.

Tento přístup má několik výhod:

- Se zařízeními je možné pracovat bez ohledu na implementaci pomocí plně virtuálního rozhraní.
- Správa všech typů zařízení je jednotná.
- Zařízení je možno identifikovat dle typu a využívat jeho specifické funkce.
- Umožňuje modulárním způsobem rozšiřovat podporu o další typy zařízení, i za běhu programu.

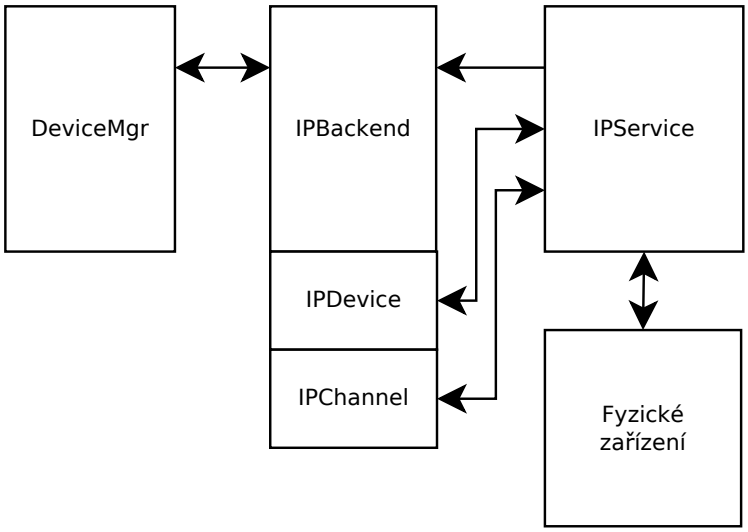
Pro potřeby implementace nových typů zařízení byly příznaky zařízení rozšířeny o následující:

- `Device::Virtual` - pokud je nastaven, jedná se o virtuální zařízení. Slouží pro potřeby uživatele odlišit fyzická zařízení od virtuálních.
- `Device::Shared` - pokud je nastaven, tak zařízení neexistuje výhradně pro danou instanci programu, ale je např. sdíleno v síti.

Kapitola 4

Sdílení zařízení v IP sítích

Sdílení zařízení je realizováno novým modulem `IPBackend`, který rozšiřuje funkcionalitu o nový virtuální typ zařízení třídy `IPDevice`. Hlavním kritériem pro výběr knihovny pro síťovou komunikaci byla jednoduchost, platformová nezávislost a nezávislost na dalších pomocných knihovnách. Z těchto důvodů jsem jako základ použil svou knihovnu liburpc, kterou jsem rozšířil o potřeby projektu. Knihovna implementuje efektivním způsobem podmnožinu standardu X.680 ASN.1[5] s kódováním X.690 BER[6] a lze ji zakompilovat přímo do modulu `IPBackend`. Modul `IPBackend` pracuje nezávisle na obecném správci zařízení, tak ukazuje schéma 4.1.



Obrázek 4.1: Blokové schéma modulu pro sdílení zařízení v IP sítích.

4.1 Modul pro sdílení zařízení v IP sítích

Modul pro sdílení zařízení v IP sítích pro knihovnu *libkitclient* se skládá z několika navzájem mezi sebou komunikujících částí.

IPDevice - implementace rozhraní třídy `Device`, poskytuje metody pro práci se vzdáleným zařízením.

IPChannel - implementace rozhraní třídy `IOChannel`, poskytuje metody pro práci se vzdáleným kanálem zařízení.

IPService - síťová služba, která zpřístupňuje zařízení připojená k dané stanici pro vzdálené klienty.

IPDiscovery - síťová služba umožňující automatické vyhledávání zařízení v místní síti pomocí metody IP multicast.

IPBackend - samotný modul pro `DeviceMgr`, který spravuje zařízení třídy `IPDevice` a sleduje jejich stav.

4.2 Implementace platformově nezávislé komunikace

Architektura protokolu sdílení zařízení je přirozeně decentralizovaná a každá připojená stanice tak může pracovat nezávisle na ostatních. Díky tomu je protokol odolný vůči výpadku jednotlivých stanic a útokům typu DoS[1] na centrální uzly. Za nevýhodu může být považována neexistence centrální databáze evidující sdílená zařízení a jejich využití. Jako nosná vrstva byl zvolen protokol TCP k zajištění spolehlivé komunikace mezi jednotlivými uzly. Komunikace se podobá metodě RPC[2], proto je nezbytné spolehlivou metodou doručit žádost a zároveň obdržet odpověď a to ve správném pořadí, jelikož odpověď může přesáhnout délku jednoho UDP paketu, zejména při blokovém čtení dat. Toto je řešitelné i protokolem orientovaným na zprávy, ale lze říci, že implementací spolehlivé metody komunikace bychom znovu došli ke spojovanému protokolu TCP.

Protože implementace síťové komunikace se liší v závislosti na použité platformě, implementoval jsem objektově orientovaný modul pro síťovou komunikaci podporující BSD sokety[13] na platformě GNU/Linux a BSD. Zároveň je podporována knihovna Windows Sockets 2[10] *(dále jen winsock2)* v prostřední Microsoft Windows. Hlavní rozdíly jsou především:

- Nutnosti inicializovat systém síťové komunikace voláním `WSAStartup()` při použití knihovny *wsock2* a rozdílné názvy pro konverzi formátu síťových adres.
- Některá systémová volání mají též pozměněnou sémantiku, např. při použití knihovny *wsock2* je třeba sokety uzavírat pomocí volání `closesocket()` které je určeno pro sokety a nikoliv voláním `close()`, které je v prostředí Microsoft Windows vyhrazeno pro popisovače souborů.
- Sémantika předaných parametrů je u některých volání rozdílná nebo musí být inicializovány na určitou hodnotu. Např. časová hodnota pro timeout při volání `select()`.

[1] Denial of Service - metoda útoku zahlcením služby, která má za následek odepření legitimních požadavků.
[2] RPC - Remote Procedure Call, metoda volání vzdálených funkcí.

Modul pro platformově nezávislou komunikaci je součástí knihovny *liburpc* přiložené ke knihovně *libkitclient* a je reprezentováno třídou `Socket`. Třída `Socket` implementuje všechny potřebné funkce pro práci se spojovanými protokoly včetně několika funkcí pro práci s jednodušším protokolem UDP. Navíc je implementována podpora pro registraci do skupin IP multicast.

4.3 Protokol pro komunikaci se vzdálenými zařízeními

Použitý komunikační protokol je založen na podmnožině standardu X.680 ASN.1[5]. Jedná se tedy o strukturovaný a především typovaný protokol. Díky širokému použití, zejména v protokolech SNMP[2] a LDAP[9], je velmi dobře dokumentovaný a v závislosti na použitém kódování i prostorově i časově efektivní. Zároveň je možné transparentně měnit kódování a např. pro komunikaci s textově založenými klienty použít kódování X.693 XER[8] založeného na jazyku XML[14]. Jako výchozí kódování jsem použil X.690 BER[6] z následujících důvodů:

- Jedná se o binární kódování a podporuje částečnou kompresi, z toho důvodu je prostorově efektivní.
- Zapisuje se přímo při vytváření PDU[3], není nutné každou zprávu kompilovat.
- Na rozdíl od X.691 PER[7] je v binární formě stále srozumitelný a čitelný.

Kódování X.690 BER

Každá datová jednotka je kódováná tzv. „tripletem“ (obrázek 4.2). První oktet reprezentuje datový typ, následující oktety délku hodnoty v bytech a poslední část samotná hodnota. V případě několika speciálních typů může být délka nulová a jednotka bude mít tedy pouze dva oktety - datový typ a nulový oktet reprezentující délku.

Typ (1 oktet)	Délka (1 - N oktetů)	Hodnota (0 - N oktetů)

Obrázek 4.2: Reprezentace datové jednotky protokolu ve formě „tripletu“.

Datový typ je reprezentován právě jedním oktetem skládajícím se z tříd, rozlišení primitivního a složeného datového typu a samotného čísla identifikující datový typ.

Třída typu je reprezentována 2 nejvyššími významovými bity v oktetu a může nabýt následujících hodnot:

- `Universal` - obecné a standardem definované datové typy, např. celé číslo, řetězec. Tyto typy se nesmí být podle standardu využité pro jiné sémantické typy.
- `Application` - vlastní typy použité v rámci jedné aplikace, lze je použít pro vytváření vlastních datových typů.
- `Context-specific` - datový typ použitý jen v určitém kontextu jako např. `SET OF` nebo `SEQUENCE`.
- `Private` - soukromé datové typy, možno použít bez omezení.

[3] Protocol Data Unit - základní datová jednotka protokolu.

P/C bit rozlišující primitivní nebo složený datový typ může nabýt dvou hodnot - pokud je jeho hodnota 1, jedná se o složený datový typ.

Číslo datového typu určuje jeho význam v dané třídě, může nabývat max. 2^5 hodnot. Pro třídu `Universal` všechny hodnoty představují přesně definovaný datový typ viz následující tabulka.

Datový typ	P/C bit	Číslo (hexadecimálně)
`BOOLEAN`	primitivní	0x01
`INTEGER`	primitivní	0x02
`OCTET STRING`	primitivní	0x04
`NULL`	primitivní	0x05
`ENUMERATED`	primitivní	0x0A
`SEQUENCE`	složený	0x10
`SET OF`	složený	0x11

Obrázek 4.3: Přehledová tabulka použitých univerzálních datových typů.

Délka je částečně komprimována a je zapsána následujícím způsobem:

- Pokud je délka obsahu menší než 128 bajtů *(0x80 hexadecimálně)*, je délka reprezentována přímo jedním oktetem o této hodnotě.
- V opačném případě první oktet reprezentuje hodnota `0x80 + N`, kde `N` představuje počet bajtů využitých pro zápis délky a samotná délka je zapsána na následujících `N` bajtech. Např. délka obsahu 1 bajt zapsaná na 16 bitech *(2 bajty)* je reprezentována jako posloupnost bajtů `0x82 0x00 0x01`. Hodnoty délky je zapsána v pořadí bajtů Big-Endian[4].
- Speciální hodnota `0x80` u strukturovaných datových typů představuje neurčenou délku datového typu. V takovém případě musí za hodnotou následovat speciální datová jednotka `EOC`. Tento způsob zápisu není v knihovně *liburpc* podporován.

Knihovna *liburpc* implementuje zápis délky nejkratším možným způsobem, tak jak to vyžadují standardy X.690 DER a X.690 CER.

0x04	0x05	't' 'e' 'x' 't' '\0'

Obrázek 4.4: Příklad kódování X.690 DER datového typu `OCTET STRING` hodnoty `"text"`.

[4]Nejvíce významný bajt se ukládá na nejnižší adresu.

Použité vlastní datové typy

Pro komunikaci mezi jednotlivými uzly bylo použito především typů třídy `Universal`, všechny přenášené zprávy jsou zapouzdřeny vlastním datovým typem třídy `Application`. Tento strukturovaný typ je podobný typu `SEQUENCE`, s tím rozdílem že nese datové jednotky různých typů. Pro zápis i čtení slouží třída `Packet`. Protože je typů zpráv méně než 2^5, je možné operační kód zakódovat přímo do datového typu zprávy. Všechny tyto datové typy jsou strukturované a náleží třídě `Application`.

Datový typ	Třída	P/C bit	Číslo (hexadecimálně)
`UNSIGNED`	`Private`	primitivní	0x02
`STRUCTURE`	`Universal`	složený	0x00
`CALL`	`Application`	složený	0x00

Obrázek 4.5: Přehledová tabulka použitých univerzálních datových typů.

Příklad práce s komunikačním protokolem

Rozhraní na práci s třídou `Packet` je velmi jednoduché, je zapotřebí:

1. Vytvořit instanci paketu s operačním kódem.
2. Přidat přenášená data.
3. Odeslat zprávu.

Na příkladu si ukážeme vytvoření zprávy s operačním kódem `DevIsOpen` s jednou datovou jednotkou typu `INTEGER` a hodnotou `7`:

```
// 1. Vytvoří prázdnou zprávu
Packet pkt(DevIsOpen); // Application class, Structured, N = 1
// 2. Přidá data
pkt.addInt8(7); // Universal class, Primitive, N = 2
// 3. Odešle zprávu
pkt.send(socket);
```

Zpráva odeslaná tímto způsobem bude mít délku `5` oktetů a obsah `0x61 0x03 0x02 0x01 0x0a`. Reprezentace oktetů je následovná:

1. `0x61` - datový typ zprávy včetně operačního kódu.
2. `0x03` - délka obsahu v bajtech.
3. `0x02` - datový typ `INTEGER`.
4. `0x01` - délka hodnoty je 1 bajt, jedná se tedy o osmibitové číslo.
5. `0x07` - osmibitová hodnota čísla[5].

[5] `INTEGER` představuje znaménkový datový typ.

4.4 Protokol pro vyhledávání zařízení v místní síti

Pro snadné sdílení zařízení v místních sítích byl navržen jednoduchý protokol pro automatické vyhledávání připojených uzlů na bázi metody IP multicast [12].

IP multicast

Metoda IP multicast doplňuje technologie unicast a broadcast. Výhodou oproti technologii broadcast je především snížení zátěže v síti a možnost vysílat data mimo přímo připojené uzly. Nevýhodou oproti technologii broadcast je potom podpora ze strany routerů na něž klade technologie multicast větší nároky, především je třeba kromě směrování pakety replikovat. Díky využití technologie multicast je tedy sdílení možno rozšířit na širší oblast, např. mezi několika učebnami a zároveň rozdělovat do skupin, což u technologie broadcast možné není.

Adresování je podobné jako v metodě unicast, ale používá se jiné třídy adres. Zatímco pro unicast se používají adresy tříd **A**, **B** a **C**, v technologii IP multicast se využívá pouze třída adres **D**[1]. V této třídě se používá pro adresování rozsah `224.0.0.0 - 239.255.255.0`, každou adresu tedy můžeme snadno identifikovat podle prvního oktetu, který v binární reprezentaci začíná hodnotou `1110`[3].
Rozsah adres `224.0.0.0 - 224.0.0.255` je rezervován pro adresování.

Dosah datagramu je možné ovlivnit nastavením hodnoty TTL *(Time To Live)*. Toto pole v datagramu IP ovlivňuje dosah paketu. Zapisuje se jedním oktetem s rozsahem `0 - 255`.

TTL	Význam
0	Pouze v rámci jednoho uzlu.
1	Pouze v rámci podsítě.
32	V rámci jedné sítě (Intranet).
64	V rámci kontinentální sítě Internet.
128	V rámci mezikontinentální sítě Internet.
255	Bez omezení.

Tabulka 4.1: Význam speciálních hodnot TTL.

Multicastová skupina je reprezentována právě jednou adresou třídy D. Jelikož jsou skupiny otevřené, je možné zasílat datagramy do skupiny bez nutnosti členství. Pro příjem datagramů je členství ve skupině nezbytné.

Implementace metody IP multicast pro vyhledávání zařízení

Pro implementaci protokolu vyhledávání zařízení v místní síti byla využita metoda IP multicast. Mimo již uvedených důvodů je výhodné i členění do multicast skupin. Je tedy možné oddělit jednotlivé uzly sdílející zařízení (např. učebny) do logických skupin. V současné implementaci je použita skupina `225.26.27.28` a port `24242` Protože technologie multicast přirozeně neposkytuje možnost spolehlivého přenosu, byl implementován jednoduchý protokol odolný vůči chybám.

Protokol pro vyhledávání zařízení

Jelikož není možné využít pro přenos dat spolehlivý protokol TCP, je nutné zaručit odolnost vůči chybám a zotavení vlastním způsobem. Na základě těchto požadavků jsem implementoval jednoduchý protokol ideální pro jakýkoliv přenos typu 1:M.

Protokol využívá pro přenos dat protokol UDP a skládá se pouze z dvou operačních kódů (viz tabulka 4.2), přičemž každý je reprezentován pouze jedním oktetem. Zároveň je protokol textový a je možné tak velice jednoduše implementovat aplikaci pouze sledující stav uzlů v síti pouhým výpisem příchozích datagramů. Každá zpráva je ukončena znakem `0x00` a je možné ji tedy do budoucna rozšířit o další metadata jako je např. symbolické jméno, počet zařízení, způsob zabezpečení aj. IP adresu ohlašovaného uzlu není třeba zapisovat do přenášených dat, přirozeně ji získáme příjmem datagramu.

Typ zprávy	Hexadecimální kód	Význam
`Announce`	`0x61 'a'`	Ohlášení připojeného uzlu.
`Leave`	`0x6c 'l'`	Odhlášení uzlu ze skupiny.

Tabulka 4.2: Typy zpráv pro ohlašování uzlů v místní síti.

- Zpráva typu `Announce` upozorní příjemce na aktivní uzel v síti a ten se může rozhodnout připojit k odesilateli spolehlivým protokolem a získat další informace.
- Zpráva typu `Leave` upozorní příjemce na odhlášení odesilatele od multicastové skupiny. To však nemusí nutně znamenat ukončení probíhající komunikace.

Protokol pro automatické vyhledávání zařízení v síti je tedy:

- Prostorově efektivní - jelikož délka přenášených dat v základní verzi je pouze 2 oktety, délka celého UDP datagramu se tedy skládá z 8 oktetů hlavičky a 2 oktetů užitečných dat.
- Velmi dobře čitelný bez nutnosti data dále zpracovat.
- Rozšiřitelný o metadata bez ztráty kompatibility.
- Odolný vůči ztrátě dat - ohlášení probíhá periodicky, ztráta dat tedy znamená oddálení zprávy o jeden interval. V případě ztráty datagramu o odhlášení uzlu ze skupiny dojde automaticky k odpojení klienta při následující aktualizaci seznamu připojených zařízení.

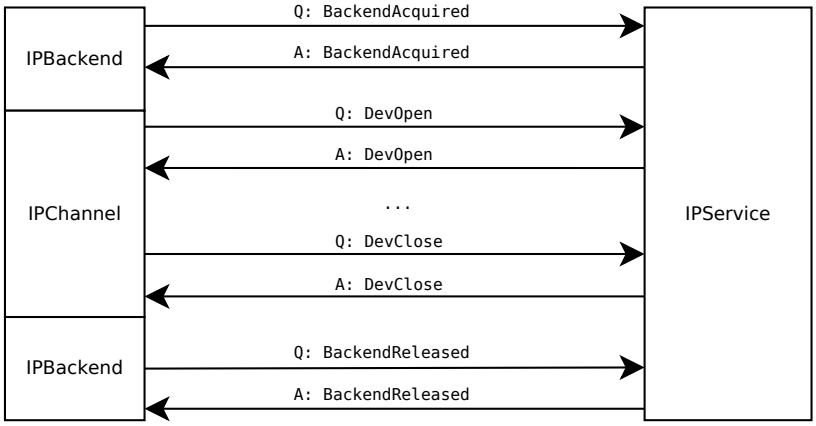
Protokol i implementaci sítovéhu uzlu představuje třída `IPDiscovery`, pro implementaci platformově nezávislé síťové vrstvy je použita vlastní knihovna popsaná v kapitole 4.2.

4.5 Komunikačního protokol v modulu IPBackend

Pro kompletní virtualizaci zařízení v kontextu knihovny *libkitclient* je třeba implementovat rozhraní tříd `DeviceBackend`, `Device` a `IOChannel`. Použitý komunikační protokol tedy musí podporovat především komunikaci mezi virtuálním a fyzickým zařízením, kde pro každé virtuální zařízení existuje právě jedno fyzické a současně také komunikaci mezi místním a vzdáleným správcem zařízení a aktualizaci stavu připojených zařízení.
Ukázka komunikace viz obrázek 4.6.

Virtualizace správce zařízení

Virtuální správce zařízení je implementován v třídě `IPBackend` a stará se o sledování stavu vzdálených zařízení. Jako nosný protokol pro virtualizaci je využit protokol na bázi X.680 ASN.1 popsaný v kapitole 3 v režimu otázka-odpověď. Díky této podobnosti protokolu s metodou RPC je možné relativně jednoduše přeložit každé volání metody modulu na právě jeden operační kód. Operační kód paketu představuje v zápisu X.690 BER strukturovaný datový typ třídy `Application` a číslem samotného kódu. Pro číslo typu je volných 5 bitů v oktetu, lze tedy tímto způsobem zakódovat 2^5 operačních kódů. V případě většího množství kódů lze použít speciální operační kód a požadovaný operační kód zakódovat jako první triplet v paketu s typem `INTEGER` následujícím způsobem:
`Packet ::= CallType { opCode INTEGER }`



Obrázek 4.6: Ukázka virtualizace správce zařízení, obsazení prvního dostupného zařízení.

Přehled virtualizovaných metod třídy IPBackend

Virtualizované metody jsou reprezentovány zprávami, přičemž každý typ zprávy odpovídá právě jedné virtualizované funkci. Formát zpráv je zapsán ve formátu X.680 ASN.1[5].

BackendUpdate - vyhodnocení seznamu sdílených zařízení. Metoda se volá periodicky při aktualizaci lokálního seznamu zařízení a slouží především k sledování stavu vzdálených zařízení. Zároveň sleduje stav spojení se vzdálenými uzly a v případě odpojeného uzlu uzavře všechny virtuální zařízení náležící odpojenému uzlu a uvolní zabrané systémové prostředky. Součástí této operace je i ohlášení uzlu v místní síti protokolem popsaným s sekci 4.4.

Dotaz

```
BackendUpdate ::= CallType + 0x10 {
   // 1 if sender expects a reply
   isQuery INTEGER(1)
}
```

Odpověď

```
BackendUpdate ::= CallType + 0x10 {
   deviceList ::= SEQUENCE {
      // Identifikátor zařízení
      deviceID INTEGER(4),
      // Identifikátor výrobce
      vendorID INTEGER(4),
      // Identifikátor prododuktu
      productID INTEGER(4),
      // Stav a příznaky zařízení
      deviceFlags INTEGER(4)
   }
}
```

Tabulka 4.3: Formát pro dotaz a odpověď operace BackendUpdate.

BackendIsDevReady - kontrola připravenosti zařízení k zahájení komunikace.

Dotaz

```
BackendIsDevReady ::= CallType + 0x11 {
   // Identifikátor zařízení
   deviceID INTEGER(4)
}
```

Odpověď

```
BackendIsDevReady ::= CallType + 0x11 {
   // Log. 1 pokud je připraveno
   boolState INTEGER(1)
}
```

Tabulka 4.4: Formát pro dotaz a odpověď operace BackendIsDevReady.

BackendAcquired - obsazení dané instance zařízení pro výlučný přístup. Jedná se pouze o upozornění, jako platná odpověď se vyhodnocuje paket se stejným operačním kódem. Obsah odpovědi se dále nevyhodnocuje.

Dotaz	*Odpověď*

`BackendAcquired ::= CallType + 0x13 {`
`  // Identifikátor zařízení`
`  deviceID INTEGER(4)`
`}`
	`BackendAcquired ::= CallType + 0x13 {`
	`}`

Tabulka 4.5: Formát pro dotaz a odpověď operace BackendAcquired.

BackendDevInitialize - inicializace zařízení po vytvoření instance. V současné verzi není vyžadována ani implementována žádná speciální inicializace virtuálních zařízení.

BackendReleased - uvolnění zařízení obsazeného operací `BackendAcquire`. Jako platná odpověď se opět vyhodnocuje paket se stejným operačním kódem jako dotaz a obsah odpovědi se dále nevyhodnocuje.

Dotaz	*Odpověď*

`BackendReleased ::= CallType + 0x14 {`
`  // Identifikátor zařízení`
`  deviceID INTEGER(4)`
`}`
	`BackendReleased ::= CallType + 0x14 {`
	`}`

Tabulka 4.6: Formát pro dotaz a odpověď operace BackendReleased.

Virtualizace zařízení třídy Device

Virtuální zařízení třídy `IPDevice` jsou vytvářena v rámci třídy `IPBackend` popsané v předchozí podkapitole. Představují tak zobrazení právě jednoho fyzického zařízení. Spolehlivý přenos dat mezi fyzickým a virtuálním zařízení je zajištěn protokolem TCP. Proti fyzickému zařízení je zde několik dalších odlišností. Z důvodu sdíleného TCP je spojení je třeba všechny operace synchronizovat a tudíž v každý časový okamžik může probíhat maximálně jedna operace. Přístup ke spojení je synchronizován v rámci celého procesu. Také stav zařízení se nezjišťuje v každém okamžiku, ale aktualizuje se periodicky v rámci metody `IPBackend::update()` z důvodu několikanásobně vyšší latence při přenosu dat přes TCP spojení. Veškerá komunikace ale probíhá formou dotaz - odpověď, pro protistranu není tedy možné upozornit tazatele na nová data.

V pojetí knihovny *libkitclient* má každé zařízení implementující rozhraní třídy `Device` až N kanálů určených ke komunikaci. Samotná instance zařízení pouze sleduje stav zařízení a jednotlivých kanálů. Tato data jsou aktualizována periodicky při každé aktualizaci správce zařízení. Můžeme využít skutečnosti, že není podstatné znát přesný stav v každém okamžiku a číst data získaná při poslední aktualizaci zařízení. Tato volání tedy není nutné virtualizovat.

Virtualizace komunikačního kanálu třídy IOChannel

Každé zařízení se skládá z N komunikačních kanálů, např. v případě zařízení s USB převodníkem FTDI FT2232 použitém v platformě FITkit se jedná o dva komunikační kanály označené písmeny **A** a **B**.

Rozlišování těchto kanálů je závislé na použitém ovladači sběrnice USB. V případě operačního systému Microsoft Windows s ovladači *D2XX* se jedná o dvě nezávislá zařízení[4] zatímco v prostředí GNU/Linux za použití knihovny *libusb* dojde k rozpoznání zařízení se dvěma komunikačními kanály.

Z toho důvodu je nutné virtualizovat zařízení až na úrovni knihovny libkitclient, která model komunikace se zařízením sjednocuje bez ohledu na použitou platformu.

Přehled virtualizovaných funkcí třídy IPChannel

Virtualizovaný komunikační kanál plně implementuje rozhraní `IOChannel`, pro získání kontextu se vždy jako první dvě datové jednotky v paketu uvádějí 32bit identifikátor zařízení a 8bit identifikátor kanálu. Každý komunikační kanál tedy lze jednoznačně identifikovat na 42 bitech. Vzhledem k tomu že identifikátor zařízení je unikátní v rámci jedné instance programu, dochází mapování identifikátorů vzdálených zařízení na místní.

DevIsOpen - zjišťuje, zda-li je komunikační kanál otevřený. Tento typ zprávy implementuje volání `isOpen()`, v případě úspěchu vrací `log. 1` v případě neúspěchu `log. 0`.

Dotaz

```
DevIsOpen ::= CallType + 0x01 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1)
}
```

Odpověď

```
DevIsOpen ::= CallType + 0x01 {
   // Odpověď
   response INTEGER(1)
}
```

Tabulka 4.7: Formát pro dotaz a odpověď operace DevIsOpen.

DevBytesIn - vrací počet nepřečtených bajtů ve vyrovnávací paměti kanálu. Zpráva implementuje volání `bytesIn()` a vrací 32 bitové celé číslo s počtem bajtů ve vyrovnácí paměti v případě úspěchu. Pokud nastane během operace chyba, vrací záporné číslo.

Dotaz

```
DevBytesIn ::= CallType + 0x02 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1)
}
```

Odpověď

```
DevBytesIn ::= CallType + 0x02 {
   // Odpověď
   bytesIn INTEGER(4)
}
```

Tabulka 4.8: Formát pro dotaz a odpověď operace DevBytesIn.

DevOpen - otevře komunikační kanál pro čtení i zápis. Zpráva implementuje volání `open()` a vrací 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`. Tento mechanismus zajišťuje kontrolu periodicky ukládaného stavu daného zařízení. Pokud je kanál již otevřen na vzdáleném uzlu a nedošlo ještě k aktualizaci jeho stavu, dojde k vrácení chybového kódu.

Dotaz

```
DevOpen ::= CallType + 0x03 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1)
}
```

Odpověď

```
DevOpen ::= CallType + 0x03 {
   // Odpověď
   openReturn INTEGER(4)
}
```

Tabulka 4.9: Formát pro dotaz a odpověď operace DevOpen.

DevClose - uzavře komunikační kanál pro čtení i zápis. Typ zprávy je inverzní k operaci `DevOpen` a implementuje volání `close()`. Výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevClose ::= CallType + 0x04 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1)
}
```

Odpověď

```
DevClose ::= CallType + 0x04 {
   // Odpověď
   closeReturn INTEGER(4)
}
```

Tabulka 4.10: Formát pro dotaz a odpověď operace DevClose.

DevStart - zahájí čtení kanálu fyzického zařízení do vyrovnávací paměti. Zpráva implementuje volání `start()` a vrací 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`. Implementace tohoto volání je závisla na použité platformě a ovladači fyzicky připojeného zařízení.

Dotaz

```
DevStart ::= CallType + 0x14 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1)
}
```

Odpověď

```
DevStart ::= CallType + 0x14 {
   // Odpověď
   startReturn INTEGER(4)
}
```

Tabulka 4.11: Formát pro dotaz a odpověď operace DevStart.

DevTerminate - ukončí čtení kanálu fyzického zařízení do vyrovnávací paměti. Zpráva implementuje volání `terminate()` a vrací 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`. Implementace tohoto volání je závisla na použité platformě a ovladači fyzicky připojeného zařízení. Operace je inverzní k `DevStart`.

Dotaz

```
DevTerminate ::= CallType + 0x15 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1)
}
```

Odpověď

```
DevTerminate ::= CallType + 0x15 {
   // Odpověď
   terminateReturn INTEGER(4)
}
```

Tabulka 4.12: Formát pro dotaz a odpověď operace DevTerminate.

DevFlush - vyprázdní vyrovnávací paměť ovladače fyzického zařízení. Tento typ zprávy implementuje volání `flush()` a výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevFlush ::= CallType + 0x05 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Příznaky pro vyrovnávací paměť
   flags INTEGER(4)
}
```

Odpověď

```
DevFlush ::= CallType + 0x05 {
   // Odpověď
   flushReturn INTEGER(4)
}
```

Tabulka 4.13: Formát pro dotaz a odpověď operace DevFlush.

DevSetBaudRate - nastaví přenosovou rychlost komunikačního kanálu. Tento typ zprávy implementuje volání `setBaudRate()` a výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevSetBaudRate ::= CallType + 0x06 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Požadovaná přenosová rychlost
   rate INTEGER(4)
}
```

Odpověď

```
DevSetBaudRate ::= CallType + 0x06 {
   // Odpověď
   baudRateReturn INTEGER(4)
}
```

Tabulka 4.14: Formát pro dotaz a odpověď operace DevSetBaudRate.

DevConfigure - operace nastaví parametry pro sériovou linku. Zpráva implementuje volání `configure()`. Jejím výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevConfigure ::= CallType + 0x07 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Počet bitů na jeden znak
   bits INTEGER(4),
   // Počet stop bitů za každým znakem
   stopBits INTEGER(4),
   // Parita
   parity INTEGER(4)
}
```

Odpověď

```
DevConfigure ::= CallType + 0x07 {
   // Odpověď
   configureReturn INTEGER(4)
}
```

Tabulka 4.15: Formát pro dotaz a odpověď operace DevConfigure.

DevSetMode - nastavení módu komunikace se sériovým zařízením. Mimo sériovou komunikaci lze přímo číst hodnoty jednotlivých pinů. Zpráva implementuje volání `setMode()`. Jejím výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevSetMode ::= CallType + 0x08 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Maska
   mask INTEGER(1),
   // Mód operace se zařízením
   mode INTEGER(4)
}
```

Odpověď

```
DevSetMode ::= CallType + 0x08 {
   // Odpověď
   setModeReturn INTEGER(4)
}
```

Tabulka 4.16: Formát pro dotaz a odpověď operace DevSetMode.

DevSetDtr - nastavení řízení toku dat způsobem DTR/DSR[6]. Zpráva implementuje volání `setDtr()`. Jejím výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`. V případě platformy FITkit dojde deaktivací obou linek řízení toku dat a následnou reaktivací k resetu mikrokontroleru.

Dotaz

```
DevSetMode ::= CallType + 0x09 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Volba
   state INTEGER(1)
}
```

Odpověď

```
DevSetMode ::= CallType + 0x09 {
   // Odpověď
   setDtrReturn INTEGER(4)
}
```

Tabulka 4.17: Formát pro dotaz a odpověď operace DevSetDtr.

DevSetRts - nastavení řízení toku dat způsobem RTS/CTS[7]. Zpráva implementuje volání `setRts()`. Jejím výsledkem je 32 bitové celé číslo s `log. 1` v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevSetRts ::= CallType + 0x10 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Volba
   state INTEGER(1)
}
```

Odpověď

```
DevSetRts ::= CallType + 0x10 {
   // Odpověď
   setDtrReturn INTEGER(4)
}
```

Tabulka 4.18: Formát pro dotaz a odpověď operace DevSetRts.

[6]Data Terminal Ready / Data Set Ready - metoda hardwarového řízení toku dat u RS232
[7]Request To Send / Clear To Send - metoda hardwarového řízení toku dat u RS232

DevRead - čtení daného počtu znaků z komunikačního kanálu. Zpráva implementuje volání `read()`. Operace vrací 32 bitové celé číslo s počtem přečtených znaků v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`. Zadaný časový limit pro operaci nezohledňuje síťovou latenci, absolutní přesnost tedy není zaručena.

Dotaz

```
DevRead ::= CallType + 0x11 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Požadovaný počet znaků k přečtení
   count INTEGER(4),
   // Časový limit na operaci
   timeout INTEGER(4)
}
```

Odpověď

```
DevRead ::= CallType + 0x11 {
   // Počet přečtených znaků
   readBytes INTEGER(4),
   // Přečtená data
   data OCTET STRING
}
```

Tabulka 4.19: Formát pro dotaz a odpověď operace DevRead.

DevReadSome - čtení znaků vyrovnávací paměti s časovým limitem. Zpráva implementuje volání `readsome()`. Operace vrací 32 bitové celé číslo s počtem přečtených znaků v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`. Zadaný časový limit pro operaci nezohledňuje síťovou latenci, absolutní přesnost tedy není zaručena. Formát zprávy je obdobný s operací `DevRead`.

DevWrite - zápis daného počtu znaků na komunikační kanál. Zpráva implementuje volání `write()`. Operace vrací 32 bitové celé číslo, které udává počet zapsaných znaků v případě úspěchu nebo záporné číslo představující chybový kód třídy `IOChannel`.

Dotaz

```
DevWrite ::= CallType + 0x13 {
   // Identifikátor zařízení
   deviceID INTEGER(4),
   // Identifikátor kanálu
   chanID INTEGER(1),
   // Požadovaný počet znaků k přečtení
   count INTEGER(4),
   // Data pro zápis
   data OCTET STRING
}
```

Odpověď

```
DevWrite ::= CallType + 0x13 {
   // Počet zapsaných znaků
   writtenBytes INTEGER(4)
}
```

Tabulka 4.20: Formát pro dotaz a odpověď operace DevWrite.

Implementace služby sdílení fyzických zařízení

Služba sdílení fyzických zařízení implementuje obsluhu požadavků vzdálených virtuálních zařízení a překládá příchozí požadavky na operace nad fyzicky připojenými zařízeními. Zároveň je třeba sledovat stav připojených uzlů a jejich vazby na fyzicky připojená zařízení. Pokud totiž dojde k přerušení spojení, je nutné uvolnit všechna zařízení obsazená daným uzlem a uvolnit systémové prostředky. Pro síťovou komunikaci je využita platformově nezávislá knihovna představená v sekci 4.2. Tato knihovna byla rozšířena pro potřeby implementace TCP serveru schopného obsluhovat více spojení v rámci jednoho vlákna s virtuálním rozhraním pro obsluhu příchozích dat.

Výsledkem třída `IPService` schopná autonomního běhu ve vlastním vlákně. Pro implementaci platformově nezávislých vláken byla použita knihovna QtCore verze 4, která poskytuje objektově orientovanou implementaci vláken v jazyce C++. Služba zpracovává jak požadavky správce virtuálních zařízení `IPBackend`, tak přímou komunikaci mezi virtuálním a fyzickým zařízením.

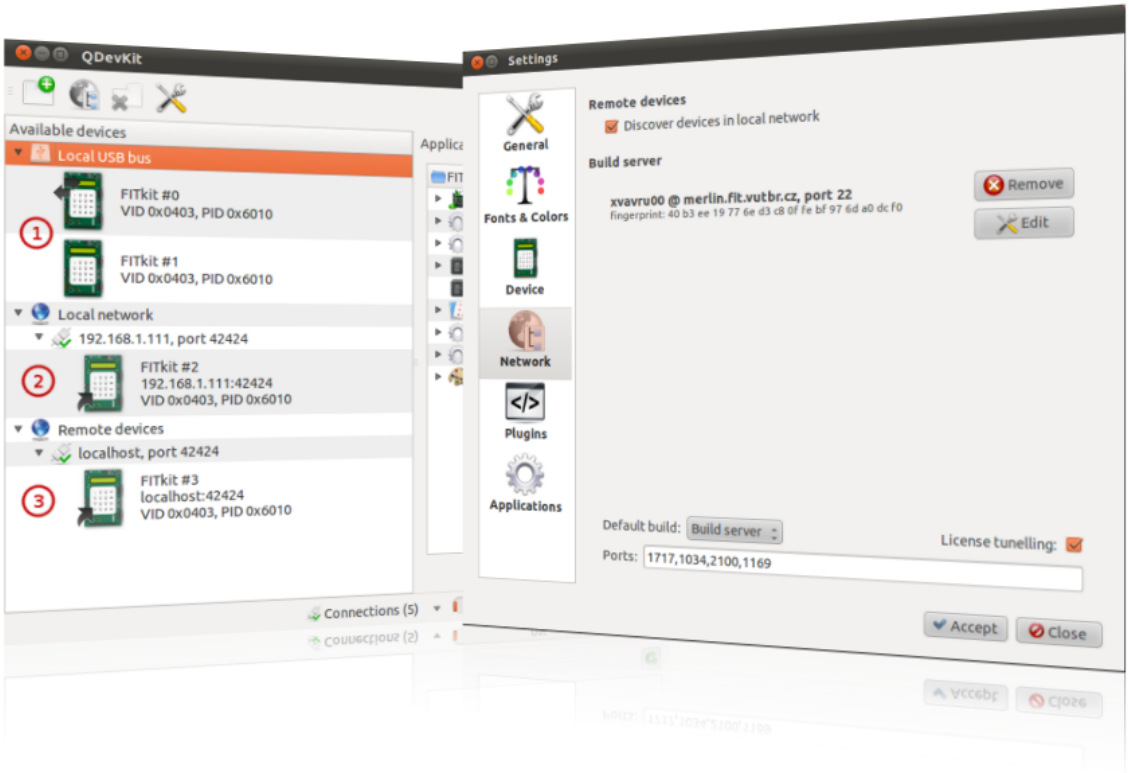
4.6 Sdílení zařízení v programu QDevKit

Prvotní motivací k mé práci bylo zpřístupnit zařízení v rámci lokální sítě, k tomu účely byla navržena a implementována podpora pro virtuální zařízení v knihovně *libkitclient* a služba automaticky vyhledávající zařízení v místní síti. Poslední logickou částí bylo zpřístupnit nové funkce uživatelům softwarového vybavení platformy FITkit ve formě grafického rozhraní. Toho bylo dosaženo rozšířením grafické terminálové aplikace QDevKit (obrázek 4.7) o následující funkce:

- Rozlišování virtuálních a fyzicky připojených zařízení.
- Možnost sdílet zařízení v místní síti.
- Zapínat a vypínat funkci automatického vyhledávání zařízení v síti.
- Zobrazovat automaticky nalezená i přímo připojená virtuální zařízení.

Virtuální zařízení jsou v aplikaci QDevKit odlišena od fyzických transparentní ikonou. Taková zařízení nelze dále sdílet, jinak je možné s nimi pracovat stejným způsobem jako s fyzickými. Jsou zároveň podporované i zásuvným modulem FKFlash a lze je tedy běžným způsobem programovat. Sdílená zařízení navíc obsahující značku symbolizující sdílení. Přehled připojených zařízení je pro větší přehlednost rozdělen (obrázek 4.7) do tří kategorií :

- Zařízení připojená ke sběrnici USB **(1)**, jedná se o fyzicky připojená zařízení, které mohou být bez omezení sdílena pro vzdálené použití.
- Zařízení automaticky nalezená v místní síti **(2)**, nalezené pomocí protokolu pro automatické vyhledávání, tato zařízení nelze dále sdílet.
- Přímo připojená zařízení v síti **(3)** - tato zařízení jsou nalezena na sběrnici USB a mohou být dále sdílena.



Obrázek 4.7: Okno aplikace QDevKit.

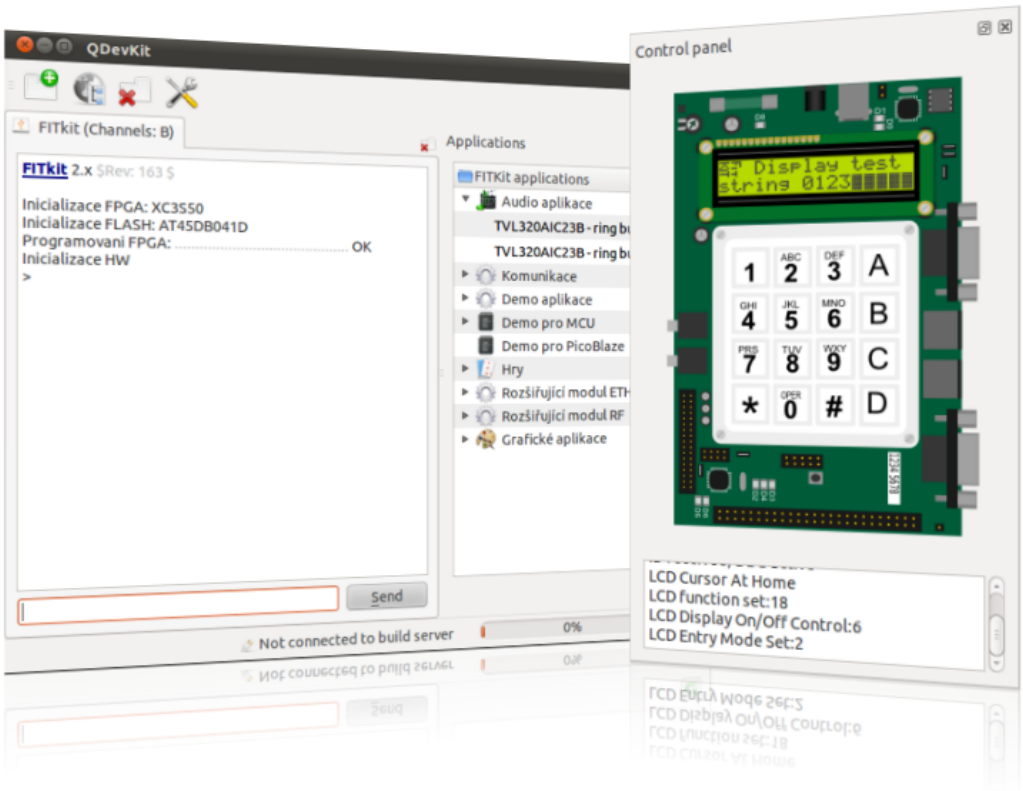
4.7 Podpora virtualizace v QDevKit

Platforma FITkit poskytuje široké spektrum periferních zařízení, které je možné využít. Jedná se především o displej, klávesnici a řadu portů dostupných v aplikacích využívajících mikrokontroler MSP430 nebo programovatelné hradlové pole Xilinx Spartan 3. Z toho vyplývá omezení práce na vzdáleném zařízení, které nemá uživatel fyzicky v dosahu. Tento problém částečně řeší zásuvný modul do aplikace QDevKit, který jsem implementoval spolu s Ing. Zdeňkem Vašíčkem a umožňuje v reálném čase zobrazovat stav displeje a přenášet stisky virtuální klávesnice. Toho bylo dosaženo implementací komunikačního protokolu v FPGA.

Díky tomu, že sériový převodník FTDI 2232C poskytuje dva sériové kanály, bylo možné volný kanál **A** využít pro komunikaci mezi zásuvným modulem a zařízením. Zásuvný modul si tak periodicky může zjišťovat stav displej a dalších periferí a zobrazovat jej v reálném čase přímo v okně aplikace.

Obsluha komunikačního kanálu běží ve zvláštním vlákně, aby nedocházelo z ovlivnění plynulosti aplikace a pro vykreslování se využívá standardní smyčka událostí knihovny Qt. To vyplývá z omezení knihovny Qt vykreslovat uživatelské rozhraní pouze v hlavním vlákně aplikace.

Zásuvný modul je napsán v jazyce C++ a je dostupný bez dalších úprav jako rozšíření aplikace QDevKit (obrázek 4.8).



Obrázek 4.8: Zásuvný modul pro zobrazení FITkitu v reálném čase.

Kapitola 5

Vzdálený překlad

Motivací pro virtualizaci překladového prostředí pro možnost překládat aplikace bez nutnosti instalovat nástroje pro překlad. Díky tomu je možné např. pracovat s FITkitem z veřejného počítače bez nutnosti instalace nástrojů pro překlad a složitého licencování. Každá aplikace pro platformu FITkit se zpravidla sestává ze dvou částí, přičemž obě vyžadují specifické nástroje na překlad.

Aplikace pro mikrokontroler napsaná v jazyce C nebo jazyce symbolických instrukcí pro rodinu MSP430 firmy Texas Instruments je součástí většiny aplikací. Zajišťuje především komunikaci s terminálovým programem a řízení programu. Pro překlad do strojového kódu je použit volně dostupný překladač MSP GCC založený na open-source překladačí GCC.

Konfigurační řetězec pro FPGA Spartan 3 představuje druhou část aplikace. Ten je případě platformy FITkit zpravidla popsaný jazykem VHDL a pro překlad se využívá vývojového prostředí Xilinx ISE, které obsahuje mimo překladač také simulátor a mnoho dalších podpůrných aplikací.

Překladač MSP GCC i vývojové prostředí Xilinx ISE jsou podporovány na platformách Microsoft Windows i GNU/Linux. Záporem je však poměrně složitá instalace a především náročnost nástrojů na prostor na pevném disku stanice, především v případě prostředí prostředí Xilinx ISE. Z těchto důvodů jsem v rámci práce navrhnul a implementoval podporu pro vzdálený překlad aplikací pro platformu FITkit bez nutnosti instalace nástrojů pro překlad.

5.1 Komunikace s překladovým serverem

Překladový server byl navržen tak, aby bylo možné překládat jak vzdáleně, tak lokálně po připojení na vzdálený terminál. Z důvodu zabezpečení kompatibilní s existujícím systémem uživatelských účtu je pro autentizaci i bezpečnou komunikaci zvolen protokol Secure Shell verze 2 *(dále SSHv2)*[15], jehož podmnožinu implementuje platformově nezávislá knihovna *libssh*. Díky použití protokolu SSHv2 je možné využít stávající systém uživatelských účtů pro přihlašování na servery FIT VUT. Knihovna nabízí více možností autentizace, z nichž jsem v práci implementoval dvě nejpoužívanější.

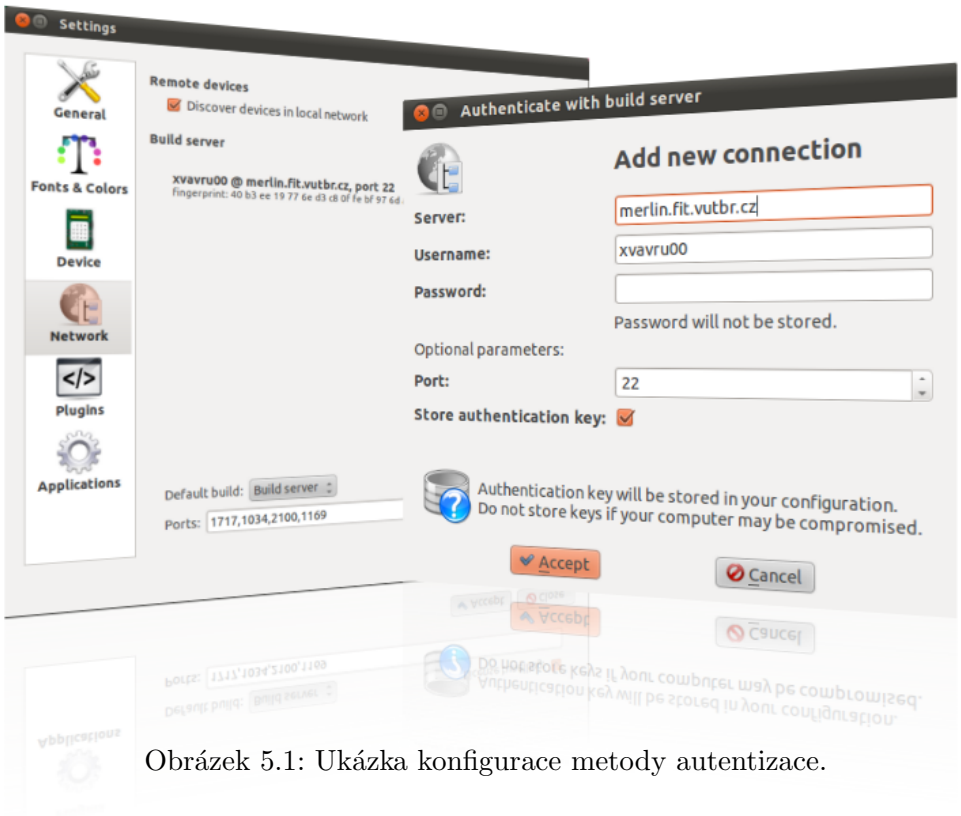
Autentizace heslem

Hlavní přednost autentizace heslem je fakt, že není třeba ukládat kryptografické tajemství na pevný disk a je tedy tedy odolná vůči ztrátě či krádeži dat. Je tedy vhodná zejména na sdílené počítače, přenosná zařízení nebo volně dostupné stanice. Určitou nevýhodou je nepohodlí vyplývající z nutnosti při každém přihlášení vyplnit heslo a také nebezpečí odezírání nebo zaznamenávaní stisků kláves. V mé práci je to částečně řešeno sdílením jednoho připojení pro komunikaci se serverem pro překlad za celou dobu běhu programu. Heslo je tedy nutné zadat pouze při prvním připojení a nikoliv pro každý překlad zvlášť. V terminologii SSHv2 se jedná o typy autentizace `password`[15].

Autentizace veřejným klíčem

Autentizace veřejným klíčem je založena na principu asymetrické kryptografie, kde existuje soukromý klíč **A** určený pro šifrování a veřejný klíč **B**, určený pro dešifrování. Klient zpravidla zašifruje tajemství svým soukromým klíčem a zašle jej serveru. Ten si jednoduše ověří tajemství tak, že se jej pokusí rozšifrovat veřejným klíčem a tak ověřit párovost soukromého a veřejného klíče. Pro tuto metodu autentizace je běžně použito šifrování DSA nebo RSA o délce klíče v řádu `1024 - 8192B`. Hlavní výhodou způsobu autentizace je pohodlnost a a odolnost vůči odezírání nebo zaznamenávání stisků kláves. Nevýhodou je potom nutnost ukládat tajemství (klíč **A**) na pevný disk stanice, kde se může stát předmětem krádeže dat. V terminologii SSHv2 se jedná o typ autentizace `pubkey`[15].

Vzhledem k tomu, že každá autentizační metoda má své výhody i nevýhody, rozhodl jsem se implementovat jak autentizaci heslem, tak veřejným klíčem. Obě volby jsou dostupné v aplikaci QDevKit (obrázek 5.1) a je možné je měnit.



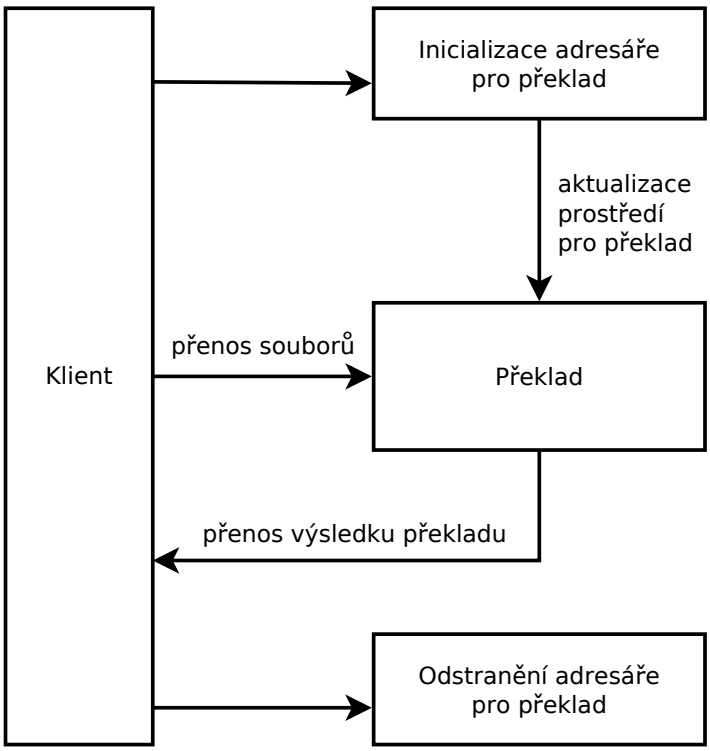
Obrázek 5.1: Ukázka konfigurace metody autentizace.

Virtualizace prostředí pro překlad aplikací

Při spuštění aplikace QDevKit s nastaveným připojením k překladovému serveru dojde k vytvoření spojení na základě námi zvolené metody autentizace. Protokol pro komunikaci s překladovým serverem se sestává s posloupnosti příkazů v jazyce SH a přenášení souborů pomocí protokolu SFTP, spojení lze tedy považovat za neinteraktivní terminál. Během životního cyklu jednoho spojení tak vzniká N kontextů pro vykonávání příkazů nebo přenos souborů. Protokol je implementován v knihovně *libfcmake*, je tedy možné jej využít i pro vzdálený překlad aplikací v terminálové aplikaci fcmake bez nutnosti instalace grafického prostředí. Toho je dosaženo buď parametrem aplikace nebo generováním speciálního *Makefile*. Zároveň je však možné využít rozhraní knihovny pro implementaci komunikace s překladovým serverem v dalších aplikacích, zejména v terminálové aplikaci QDevKit.

Protokol pro práci s překladovým serverem

Samotný překladový server je implementován ve formě souboru skriptů v jazyce SH. Pro korektní funkci je nutné mít na serveru zprovozněné prostředí pro překlad aplikací pro platformu FITkit, tedy zejména překladač MSP-GCC a vývojové prostředí Xilinx ISE. Samotný překlad aplikace se sestává z více nezávislých kroků (obrázek 5.2).



Obrázek 5.2: Posloupnost operací v protokolu pro vzdálený překlad aplikací.

Inicializace adresáře pro překlad aplikace je prvním krokem, kdy je pro každého uživatele vygenerován adresář pro překlad v adresáři pro dočasné soubory. Současně je aktualizován aktuální strom aplikací platformy FITkit a nastaveno prostředí pro překlad aplikace. Tento krok obstarává skript `bserver-prepare` a jeho výstupem je cesta k adresáři pro překlad. Ta se zkládá z cesty k adresáři pro dočasné soubory, uživatelské jména a náhodného čísla. To je nutné především proto, aby mohlo probíhat více paralelních překladů pro jednoho uživatele.
Náhodná čísla adresářů jsou generována tak dlouho, dokud se nenajde první neobsazené číslo. Příklad takové cesty je např. `/tmp/xvavru00/build.12345`.

Aktualizace souborů potřebných pro překlad aplikací je druhý krokem. V případě práce přímo na překladovém serveru je nutné do připraveného adresáře nahrát všechny potřebné soubory. Pokud překlad probíhá vzdáleně za pomocí aplikace fcmake nebo QDevKit, tak se o vyhledání a přenos všech potřebných souborů stará aplikace. Díky přechodu překladového systému aplikací platformy FITkit na formát XML, je možné v rámci knihovny *libkitclient* sestavit strom závislostí a přenášet tak pouze potřebné soubory pomocí protokolu SFTP.

Překlad aplikace probíhá pomocí skriptu `bserver-make`, který má za úkol nastavit prostředí a zavolat nainstalované nástroje pro překlad. V tomto okamžiku probíhá překlad stejným způsobem, jako v případě překladu na lokálním počítači. Vzdálený překlad v rámci aplikací fcmake a QDevKit zachytává výstup terminálu v reálném čase a interpretuje jej uživateli. Po skončení skriptu je propagován návratový kód aplikace stejným způsobem jako v případě překladu na lokálním počítači.

Zpracování přeložené aplikace představuje poslední krok překladu. V případě úspěchu je nutné přenést zpět výsledek překladu pomocí protokolu SFTP a poté smazat adresář pro překlad aplikace. Pokud uživatel překládá aplikaci přímo na vzdáleném serveru bez využití aplikací fcmake a QDevKit, je doporučeno po ukončení práce smazat použité adresáře pro překlad.

Vzdálený překlad v aplikaci fcmake

Převážná část komunikace s překladovým serverem je implementována v rámci sdílené knihovny *libfcmake*. Klíč pro autentizaci je realizován třídou `Key` a jejich správu obstarává třída `KeyChain`. Metoda autentizace heslem nevyžaduje žádnou další implementaci. Samotné řízení spojení se serverem je implementováno třídou `Remote` a zajišťuje především autentizaci metodou zadání hesla nebo veřejným klíčem. Pár klíčů je generován na straně serveru vzhledem k tomu, že je vyžadován běh knihovny *libfcmake* na více platformách a knihovna *libssh* tuto funkcionalitu neposkytuje. Zároveň je poskytováno rozhraní pro obousměrný přenos souborů pomocí protokolu SFTP a možnost vykonávat vzdáleně příkazy ve formě neinteraktivního terminálu. Toho je využito pro funkce pro přípravu překladového adresáře a samotné funkce pro překlad.

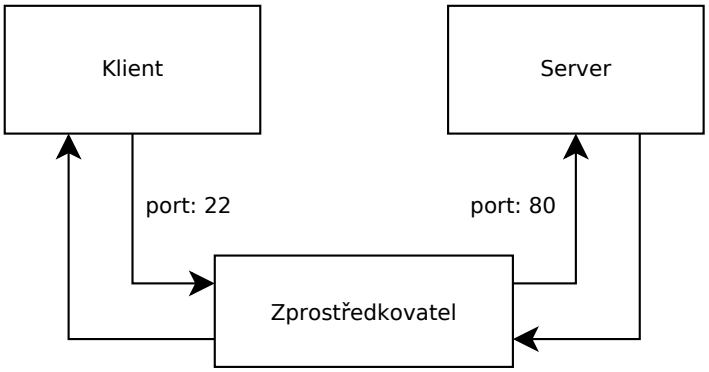
Samotná aplikace fcmake umožňuje jak vytvořit *Makefile* pro vzdálený překlad pomocí parametru `--remote`. Alternativně lze použít parametr `--remote-build` a vykonat tak samotný vzdálený překlad přímo v aplikaci fcmake bez nutnosti generovat *Makefile*.
Formát příkazu je následující: `fcmake -r xlogin00@server`.
Adresa překladového serveru musí být dosažitelná (nevytváří se tunel jako v případě spojení z aplikace QDevKit).

5.2 Tunelování spojení na překladový server

Pro účely překladového serveru nám byla poskytnuta stanice vyhrazená pro vzdálený překlad aplikací pro platformu FITkit. Stanice je díky osazení procesory Intel Xeon E5640 a 48GB operační paměti dostatečně dimenzována pro obsluhu paralelně běžících žádostí o překlad aplikací, především na náročnou syntézu konfiguračního řetězce pro programovatelné hradlové pole. Dostupnost stanice je však omezená pouze v rámci sítě VUT. Z toho důvodu bylo nutné navíc implementovat systém tunelování spojení přes veřejně dostupný server v síti VUT a zajistit tak funkčnost vzdáleného překladu i pro uživatele mimo tuto síť. Protože protokol Secure Shell implementuje i možnost tunelování portů[15], rozhodl jsem se využít stávající knihovny *libssh* k implementaci platformově nezávislé služby pro síťové tunelování.

Síťové tunelování v protokolu Secure Shell

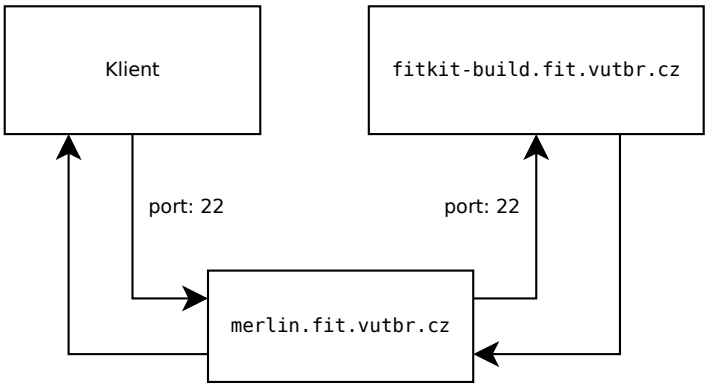
Síťové tunelování je metoda zapouzdření původního síťového protokolu jiným. Je možné jej implementovat jak na linkové úrovni modelu ISO/OSI, kde je typickým představitelem protokol Layer 2 Tunneling Protocol *(L2TP)*, tak pro potřeby mé práce zajímavější aplikační vrstvě. Síťové tunelování popsané protokolem Secure Shell je pro potřeby mé práce zajímavé hned z několika hledisek.



Obrázek 5.3: Příklad tunelování spojení protokolu HTTP přes prostředníka.

Výhoda společná pro jakýkoliv protokol síťového tunelování je obcházet omezení aktivních síťových prvků. Díky této vlastnosti je možné např. tunelovat jinak blokovaný protokol HTTP přes vzdálený server (obrázek 5.3). Konkrétním využitím pro mou práci je tunelování spojení protokolu Secure Shell na překladový server přes veřejně dostupný server v síti VUT (obrázek 5.4).

Druhou vlastností síťového tunelování v protokolu Secure Shell je přirozeně šifrovaný přenos. Veškerá komunikace je tedy přenášena po bezpečném kanálu, toho je využito pro možnost tunelovat další spojení. Zabezpečení přenosového kanálu je využito pro možnost tunelování spojení k serveru poskytujícímu licence pro simulační prostředí ModelSim.



Obrázek 5.4: Tunelování spojení na překladový server přes merlin.fit.vutbr.cz

Popsanou metodu síťového tunelování je možné popsat i jako formu přesměrování portů. Představuje tedy virtuální spojení mezi dvěma síťovými prvky, mezi nimiž funguje jeden jako prostředník umožňující jejich propojení. Cílový síťový prvek musí být pro prostředníka přímo dosažitelný, v případě takové situace je možné síťové tunely řetězit. Komunikace mezi nimi probíhá zabezpečeně, v rámci jednoho kanálu spojení protokolu SSHv2.

Implementace vzdáleného překladu v aplikaci QDevKit

Aplikaci QDevKit jsem rozšířil o službu implementující síťové tunelování za pomocí knihovny *libssh*. Protože se jedná o aplikační vrstvu, je nutné neustále obsluhovat síťový provoz na vytvořených spojeních a sledovat nové požadavky o připojení. Zároveň je nutné zajistit aby obsluha probíhajících spojení neblokovala hlavní smyčku událostí knihovny Qt a nedocházelo tak ke zpomalování aplikace. Toho bylo dosaženo vícevláknovou implementací obsluhy probíhajících spojení. Pro každý vytvořený síťový tunel je vytvořena instance třídy `TunnelService`, která využívá vlastnosti třídy `QTcpServer` asynchronním způsobem zpracovávat příchozí spojení. Pro každé vytvořené spojení je pak spuštěna obsluha ve zvláštním vlákně.

Služba pro síťové tunelování rozlišuje tunel k překladovému serveru a ostatní volitelné tunely. Ty jsou využity především k tunelování spojení k licenčnímu serveru. Protože se seznam portů může kdykoliv změnit, je možné je konfigurovat formou seznamu tunelovaných portů. Ty je možné změnit v nastavení aplikace QDevKit na kartě *Síť*. Zároveň je možné je aktualizovat automaticky formou zásuvných modulů v jazyce Python i C++.

Existující síťové tunely sdružuje třída `TunnelService`, která navíc zobrazuje stav síťových tunelů a spojení s překladovým serverem formou tlačítka ve stavové liště programu. V případě ztráty spojení je možné se stisknutím znovu připojit.

Správa spojení k překladovému serveru

Správa spojení k překladovému serveru umožňuje odlišit tunelované spojení od přímého a umožňuje tak využít tunely, které si již uživatel vytvořil před spuštěním aplikace. V případě automatického tunelování přes server `merlin.fit.vutbr.cz` je vytvořen tunel z portu `20022` na místním počítači na překladový server. Spojení zprostředkovává server `merlin.fit.vutbr.cz`, viz obrázek 5.4. Při žádosti na vzdálený překlad je tedy možné ověřit stav spojení a požadavek případně odmítnout. Zároveň uchovává spojení k překladovému serveru po dobu běhu aplikace. Není tak nutné při každé žádosti o vzdálený překlad zadávat heslo nebo se vícekrát autentizovat pomocí veřejného klíče.

Konfigurace připojení k překladovému serveru s vlastním tunelem

Aplikace QDevKit i FCMake umožňují připojení na překladový server přes vlastní tunel. Je nutné jej vytvořit tak, aby koncový bod byl vždy překladový server s adresou `fitkit-build.fit.vutbr.cz`, např. za pomocí aplikací OpenSSH nebo Putty v prostředí Microsoft Windows. Pomocí OpenSSH se takový tunel vytvoří následujícím příkazem:
`ssh -L 20022:fitkit-build.fit.vutbr.cz:22 xlogin00@merlin.fit.vutbr.cz -N`

Vytvořený tunel je možné využít v aplikacích QDevKit a FCMake tím způsobem, že jako adresa překladového serveru se nakonfiguruje `localhost` a port `20022`. Popsaným způsobem je možné vzdáleně překládat z aplikace FCMake i mimo síť VUT.
Port `20022` je možné nahradit jiným dostupným.

Tunelování spojení na licenční server

Pro správnou práci simulačních a překladových nástrojů je zapotřebí platná licence. V rámci síťě VUT je dostupná na licenčním serveru. Pro práci z domu nebo mimo síť VUT však server není dostupný. Řešením, které jsem navrhnul a implementoval je rozšířit systém tunelů na licenční server. Protože se však seznam portů licenčních služeb nepravidelně pravidelně mění, ve spolupráci s Ing. Zdeňkem Vašíčkem jsem navrhnul rozšíření zásuvného modulu pro aplikaci QDevKit, který automaticky aktualizuje seznam tunelovaných portů.

Pro správnou funkci je třeba v souboru `hosts` nastavit mapování doménového jména `semik` na adresu `127.0.0.1`. A nastavit proměnné určující adresy licenčních služeb `LD_LICENSE_FILE=1717@localhost` a `XILINXD_LICENSE_FILE=2100@localhost`.
Správnou funkci lze ověřit pomocí utility `lmutil lmstat`.

Kapitola 6

Závěr

Cílem bakalářské práce bylo rozšířit softwarové vybavení platformy FITkit o možnost sdílet zařízení v IP sítích za účelem propagace, realizace virtuální laboratoře a krátkodobého půjčování FITkitu mezi studenty. Současně také řešit problém náročné instalace softwarových nástrojů pro překlad aplikací pro FITkit a nedostupnost licencí mimo síť VUT. Pro možnost sdílených zařízení je knihovna *libkitclient*, která poskytuje jednotné rozhraní pro komunikaci s FITkitem, rozšířena o podporu virtuálních zařízení a možnosti spravovat více typů zařízení současně. Síťová komunikace je zajištěna vlastní platformově nezávislou knihovnou a binárním komunikačním protokolem implementujícím standard X.680 ASN.1 s kódováním X.690 BER. Na těchto základech je vytvořen modul `IPBackend`, který umožňuje pracovat se vzdálenými zařízeními a současně automaticky vyhledávat nová zařízení v místní síti. Aplikace QDevKit je rozšířena o uživatelské rozhraní pro práci se vzdálenými zařízeními, a také o zásuvný modul pro vizualizaci periferií FITkitu vytvořený ve spolupráci s Ing. Zdeňkem Vašíčkem. Problém náročné instalace nástrojů pro překlad aplikací řeší podpora vzdáleného překladu. Komunikace s překladovým serverem je zabezpečená pomocí protokolu SSHv2 s podporou více způsobů autentizace. Klientská část sestává z aplikace FCMake, která umožňuje vzdálený překlad z terminálu, a především uživatelského rozhraní aplikace QDevKit. Současně je v aplikaci QDevKit implementována služba pro tunelování spojení a řeší tak problém nedostupnosti překladového a licenčního serveru mimo síť VUT.

Protokol je možné dále rozšířit o podporu obousměrné komunikace, a částečně tak eliminovat síťovou latenci pozorovatelnou při znakovém čtení dat. Další možností rozšíření komunikačního protokolu je jednoduché zabezpečení sdílených zařízení, např. formou číselného PINu nebo hesla. Protokol pro vzdálený překlad je také možné rozšířit pro jiné typy platforem.

Virtualizace prostředí pro překlad aplikací byla ověřena v letním semestru 2011 v kurzu INC, kde vzdálený překlad využilo přibližně 125 studentů.

Literatura

[1] Albanna, Z.; Almeroth, K.; Meyer, D.; aj.: IANA Guidelines for IPv4 Multicast Address Assignments. 2001.

[2] Case, J. D.; Fedor, M.; Schoffstall, M. L.; aj.: Simple Network Management Protocol (SNMP). 1990.

[3] Filip, O.: Úvod do IP multicastu. `http://www.lupa.cz/clanky/uvod-do-ip-multicastu` [cit. 2011-04-03], 2004.

[4] FTDI: Software Application Development - D2XX Programmer's Guide. Future Technology Devices International Ltd. 2011.

[5] ITU-T Rec. X.680: Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. `http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf` [cit. 2011-03-16].

[6] ITU-T Rec. X.690: Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). `http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf` [cit. 2011-03-16], 2002.

[7] ITU-T Rec. X.691: Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER). `http://www.itu.int/ITU-T/studygroups/com17/languages/X.691-0207.pdf` [cit. 2011-03-16], 2002.

[8] ITU-T Rec. X.693: Information technology – ASN.1 encoding rules: XML Encoding Rules (XER). `http://www.itu.int/ITU-T/studygroups/com17/languages/X.693-0112.pdf` [cit. 2011-03-16], 2001.

[9] Koutsonikola, V.; Vakali, A.: LDAP: Framework, Practices, and Trends. *IEEE Internet Computing*, ročník 8, 2004: s. 66–72, ISSN 1089-7801, doi:10.1109/MIC.2004.44.

[10] Microsoft: Windows Sockets 2. `http://msdn.microsoft.com/en-us/library/ms740673(v=vs.85).aspx` [cit. 2011-04-05].

[11] Molkentin, D.: *The book of Qt 4 : the art of building Qt applications.* San Francisco: No Starch Press, 2007.

[12] Quinn, B.; Almeroth, K.: IP Multicast Applications: Challenges and Solutions. 2001.

[13] The IEEE and The Open Group: The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 Edition. `http://pubs.opengroup.org/onlinepubs/009695399` [cit. 2011-04-03], 2004.

[14] W3C XML Working Group: *The XML 1.0 Standard*. Network Theory Ltd., páté vydání, 2010.

[15] Ylonen T., L. C.: The Secure Shell (SSH) Authentication Protocol. 2006.